BOLDFIELD LIMITED
COMPUTING

DATABASE

Users Manual

INTRODUCTION

This DATABASE program is a very Flexible piece of software designed to allow the user to create his own database files which can store, retrieve, modify and process data to his own requirements.

It is user programmable, allowing data to be both numeric or text, and has many outstanding features.

The manual is presented in two parts. The first section deals with the simple commands and functions already built into the software, while the second section shows how to extend these facilities to enable even more powerful operations to be undertaken.

LOADING INSTRUCTIONS
To load the program enter: load database

SECTION ONE - THE SIMPLE COMMANDS AND FUNCTIONS

When you first load DATABASE it contains a demonstration file all about the countries in Europe, and we shall use this to illustrate the various commands.

If you wanted to keep this data in a conventional card system, you would probably have a separate card for each country, and on each card you would have printed the Following headings: Country, Area, Population, Capital City, Currency, and Notes. Now this same idea is used in DATABASE, but instead of calling each entry a card, it is called a RECORD, and the headings are called FIELD NAMES. Records start at number 1 (record number 0 is special, see later), and fields start at number 0. Other than the restriction of available memory, there is no limit to the number of fields in a record, and no limit to the number of records in the file.

IMPORTANT NOTE: Throughout this manual commands are shown in upper case (capital letters), but this is only for clarity in presentation, and you can actually enter them in either upper or lower case.

BROWSING

So, lets have a browse through Europe by entering a few commands:

FIR	displays the FIRst record	(Albania)
LAS	displays the LASt record	(Yugoslavia)
BAC	steps BACkwards one record	(Vatican)
FOR	steps FORward one record	(Yugoslavia)
DIS	Displays current record	(Yugoslavia)

OK, now enter FIR again (Albania), and then try entering: REP FOR (Andorra) and every time you press ENTER the program Reheats the Forwards instruction (Austria, Belgium, Bulgaria, etc..). REP is useful in conjunction with BACK, FOR, and LISP (see below).

To abandon a sequence of Reheating, simply enter: A which stands for Abort and also wipes the stack clean. In fact if you encounter the end of the file (known as OF) by attempting to go beyond either end of the records, or by requesting a non-existent record, an automatic abort is carried out.

LISP is a particularly good command for jumping around in the database, and works by reading the top number from the normal FORTH stack, makes the corresponding record current, and displays it. In it's simplest form you could just enter: 12 LISP (Greece). But better still, try: 12 14 7 REP LIS (Denmark) - the 7th record is seen first, and when you press ENTER a couple more times you will see the 14th and the 12th records (Iceland, Greece).

Because LISP reads from the stack, you can do things first to generate the numbers of the records you want to see. Some of the best facilities in DATABASE do exactly that, but you can also invent your own! For example, and completely pointlessly, you could enter 2 2 + LISP in order to see the 4th record (Belgium).

SCREEN LAYOUT

There are several commands which control what is displayed, and how it looks on the screen. So far you have been seeing one record at a time with all the field names separated from the field data by small dots, and a clear screen between consecutive records.

Now, if your records are too long to see on a single screen, there is an option to pause the display between fields and to continue only after pressing ENTER. This action is achieved by entering: 1 STEP
(revert with 0 STEP)

The tab setting for the field data can be adjusted to any column between 1 and 31, with automatic overflow to the next line. The command is: n TABS If n is entered with a preceding minus sign, the field names are suppressed. (revert with 15 TABS)

The records are normally displayed with a blank line between each field, but this can be omitted by entering: 0 GAP (revert with 1 GAP)

If the field names are shown (see TABS above), the character which is seen separating them From the data can be changed to any ASCII value by entering: n DOTS This is initially 95 DOTS (small dots), but some users may prefer spaces (32 DOTS) or dashes (45 DOTS) etc.

An alternative to clearing the screen between records, is to have the records displayed in a scrolling style - this is particularly useful when using a printer. The command is: n SCROLL where n is between 1 and 23 and is the number of blank lines between consecutive records.
(revert with 0 SCROLL)

If you don't wish to display all the fields in each record, you can select which field number to start with, and which to finish on. The form of the command is n m PART where n is the start number and m is the finish number. If you prefer, m can be entered as -1 which means finish on the last field. (revert with 0 -1 PART)

So, say that you wanted field names suppressed, the data to be tabbed along four spaces, no gaps between the lines of data, you did not want to include the NOTES, and because you were printing the results you wanted 4 line feeds between records - then you would enter:

-4 TABS 0 GAP 4 SCROLL 0 4 PART

These settings will remain in force until you change them.

(the original settings were: 15 TABS 1 GAP 0 SCROLL 0 -1 PART)

LOOKING FOR WORDS

This DATABASE package does far more than just hold the data and display it in various clever ways! One of the most useful operations is to be able to ask the computer to look through all the records and to find those that comply with some requirement that you have specified.

The simplest thing would be to, say, find and display the record relating to France. To do this you use the interrogation command ?
Try: ? France (NOTE: you must only have one space between ? and France, as other spaces would be deemed to be part of the word to be found)
You can specify the search word either in Full or abbreviated, but there are some rules! What you tell the computer to find must exactly match the data as held in the record, so FRANCE (all in capital letters) would not be found. You can abbreviate to any degree you like, and the computer will find the first occurrence of the entry. So ? F would Find Finland, and ? Fr would find France. The computer always looks from the beginning of the data File when you enter the command in full (so to do ? F followed by another ? F would just find Finland twice), but you can continue a search maintaining the original criteria by simply entering another ? symbol- so ? F (finds Finland) ? (finds France).

Now these searches have all been related to the first field name (COUNTRY) but you can search on any field name you like by making it the current one. To change you use the command BY Eg: BY CAPITAL ? Rome
which finds Italy. You can abbreviate the field name if you wish, so BY CA is the same as BY CAPITAL, but BY C would be taken as BY COUNTRY because COUNTRY would be encountered before CAPITAL. An important point to remember is that the field name (or it's abbreviation) must be entered correctly, or the computer defaults to a global search and looks in all the field names to find a match - it also switches the computer into 2 MODE (see below). This facility is often required, so you could deliberately enter a nonexistent field name to select this wider searching ability.
Eg: BY * ? Rome would still produce Italy, but the computer has looked on every data field to find it!

So far we have only considered looking for words (or strings of characters) which have been entered in full, or abbreviated by omitting some of the right hand side, but there are more powerful ways of treating them, and these are set by entering one of four MODES.

0 MODE sets the computer to look for unabbreviated strings
1 MODE allows omission of some of the right side (like LEFT$ in BASIC)
2 MODE allows both left and right omissions (like MID$ in BASIC)
3 MODE allows omission of some of the left side (like RIGHT$ in BASIC)

(When first loaded, DATABASE assumes 1 MODE)

So, with the CAPITAL field as the current one (BY CA), we could test some MODEs. Enter: 0 MODE ? Rome and Italy is found, but: ? Rom produces the message 'NOT IN FILE'. Enter: 1 MODE ? Rom and Italy is found, ? Ro produces the same result, but ? R now finds Iceland because it's capital Reykjavik is encountered before Rome. Enter: 2 MODE ? om and Italy is found (because om is a central abbreviation of Rome), but: ? o now finds Andorra because this is encountered first.

The quotation symbol (") has two functions. First, it can be used to include trailing spaces in a search word. For example, say you had some field data which looked like ... Part no. 345 56 12 ...This could be found by abbreviating in, say, 2 MODE to: ? 56 But if this was not sufficiently unique, you could enter: ? 56 " which now includes the two spaces before the 12. The second function is in separating multiple instruction on the same line after using an interrogation command.
Eg: 1 MODE BY CO ? R" BAC which finds Portugal (the country before the first one beginning with an R).

? has beer very useful in finding the first occurrence of any piece of data, but isn't the best way of finding repeat matches. Another command, ?? has been included to meet this requirement, and it works in a different way. Instead of displaying the records that match your chosen criteria, it writes the record numbers into the FORTH stack, and tells you how many were found.
You can then display the results, if you want, by REP LIS followed by the appropriate number of ENTERs. Alternatively, you can see the record numbers directly by using the command .5 which non-destructively prints the stack contents.
Eg. (assuming that you have done the last example, and that the computer is in 1 MODE, with COUNTRY as the current field name): ?? A will find all the countries that begin with an A. The result is '3 RECORDS FOUND', and you can see the record numbers by entering: .S (1,2,3) or the records themselves by REP LIS and 2 ENTERs.

To be really impressive, ?? can also work in conjunction with two more commands #AND and *OR, to any degree of complexity. So, for example, you could interrogate the database to tell you of all the records that contain both the ending 'land' to the country name, and that have the letter 'r' in the capital city name

```
3 MODE BY CO ?? land" 2 MODE BY CA ?? r" #AND                    (23,29)
```

Note how AND listed the stack contents (similar to .5). To see the records enter REP LIS (Switzerland) and press ENTER (Poland). Now this example missed out cities beginning with an R, because we only asked for lower case 'r' . So, what we need to do is include a #OR command:

```
2 MODE BY CA ?? r" ?? R" #OR 3 MODE BY CO ?? land" #AND         (14,23,29)
```

This time we get Switzerland, Poland and Iceland when we look at the results. DATABASE allows a single level of bracketing in these instruction lists, so this example could have been entered as:

```
3 MODE BY CO ?? land" { 2 MODE BY CA ?? r" ?? R" #OR }  #AND
```

In these three examples, you would have seen on the screen how the computer arrives at the results by doing the work in stages. The intermediate answers to each interrogation are placed on the stack, and the special commands either eliminate stack entries that only occurred once (#AND), or remove any stack entry that is repeated (#OR).

The last command to consider in this section is NOT. Quite simply this is used in front of ? or ?? to reverse the meaning.

So: 2 MODE BY CO NOT ?? e" .S finds 17 countries without an 'e' in the name.

DEALING WITH NUMBERS

Numbers are held in the database in exactly the same way as words, and that is just as a string of characters. So all the commands you have met up to now are equally valid for working on numbers. Eg: 0 MODE BY AREA ? 324,219 finds Norway.

However, because many uses of DATABASE will benefit from numeric comparisons, there are some additional commands to provide these facilities. The first is ?< that finds numbers less than a specified value (naturally, NOT ?< would find numbers equal to or greater than the value), and displays the first encounter of a matching record.

?(324,220 finds Albania, ?< finds Andorra (the second occurrence) etc.

As you might expect there is another command ??< that finds all the matching records and leaves the record numbers on the stack.

??< 324,220 finds 29 countries with areas less than 324,220 sq. km.

You can combine multiple instructions on a single line to interrogate the database for various conditions. Eg: NOT ??< 300,000 ??< 600,000 #AND will find 7 countries with areas in the range 300,000 to 600,000 sq.km.
Incidentally, if you have been following all the examples and not LISting the answers the stack will now contain lots of earlier results! So here is a tip. Empty the stack (A for Abort does this) before any operation where you will want to use the resulting record numbers.

Initially, DATABASE assumes the number of decimal places is zero, and it also ignores commas (marking the thousands and millions) when dealing with numbers. So: ??(324219.99 is the same as ??(324,219 which does not find Norway amongst it's 28 matches. You can change the number of decimal places by storing a new value to a system variable known as DPS (see section 2 of the manual for further details).

Mixing of numeric and text commands is allowed, so if you wanted to find all the countries that had areas between 300,000 and 400,000 sq.km. and which were also republics (as mentioned in the NOTES field), you could enter: 2 MODE BY AR NOT ??< 300000 ??< 400000 #AND BY NO ?? pub" #AND
the answer is Poland, Italy and Finland. Notice how we tested for a republic by asking for pub". This was a calculated risk (!) to include records that had the word 'Republic' (ie. the first word in the NOTES, beginning with a capital letter) as well as the word 'republic' (as it would appear elsewhere in the notes). We would of course come unstuck had one of the notes said 'lots of pubs here'! Another tip - this type of problem is avoided if all your data is always entered in one form, such as upper case. Another point is that the special numeric commands ?< and ??< look at the whole number (as if 0 MODE had been entered) so it did not matter that the entire instruction was apparently handled in 2 MODE.

SORTING DATA

It is often very useful to sort the records into some regular order. The countries of Europe are already arranged alphabetically, but now lets change it s, that the records are in capital city reverse alphabetical sequence. Enter: BY CA 0 1 SORT this takes about 20 seconds to do and ends up with Poland as the first record (capital Warsaw).
The form of the sort command is: BY Fieldname n m SORT where fieldname is the valid expression for an existing field name, n is 0 for an alphabetic sort or 1 for a numeric sort, m is 0 for ascending order or 1 For descending order.

ADDING, DELETING AND ALTERING

To add a new record to the file, simply enter the command ADD and you will be prompted by the computer to enter each item of field data in turn. The new record will be added immediately after the record which was current when you entered ADD. This is fine except for one instance, and that is if you wanted to ADD the new record as the first one in the file. In this case you must enter 0 LIS (which displays the special 0th record) before ADD.

To delete a record, simply make it current and enter DEL (it is not possible to delete the 0th record).

To alter data in any field of any record, you must first make the record current, and then enter ALT followed by the field name. For example, say you have just given birth to 1,000 new babies and you wanted to modify the population data of the UK. Enter: 32 LIS ALT POP this will first make the 32nd record current, and then position the POP field data (56,009) in the input area of the ACE display, and prompt you to alter it. Now simply adjust the data to 56,010 and press ENTER. Notice how the field name can be abbreviated (P alone would have been sufficient), but be careful to enter this correctly because invalid names will result in the computer defaulting to modify the data in the first field.

In addition to these facilities, you can also add, delete and alter the fields themselves. To do this you must operate on the special 0th record, which you make current by entering: 0 LIS . To add a new field between CAPITAL and CURRENCY called MAIN EXPORT, you would enter: BY CAP ADD MAIN EXPORT . This has the effect of adding the new field name (without any data) to every record in the file. Had you wanted this as the first field, you would hale used a non-existent field name after BY (eg: BY A ADD MAIN EXPORT). To delete fields simply enter, for example: DEL NOTES (with 0th record current), and all the NOTES data is removed from every record. For your own protection, DATABASE will display an ERROR warning if the field name you entered does not exist. Eg: DEL Capital produces the warning.

To alter the field names enter: ALT CAP for example, and CAPITAL appears in the input area for modification. If you changed this to CAPITAL CITY (press ENTER) all the records will now be modified with the new heading.

A point to remember when entering field names is that the command BY only reads up to the first space in a Field name. So although you are allowed to have spaces (as in MAIN EXPORT), BY would not be able to tell this apart from say MAIN IMPORT . Consequently, either use different first words, or use something other than a space (eg: MAIN-EXPORT).

CREATING NEW DATABASES

Unless you bought DATABASE just to find out about the countries of Europe, you will most likely want to scrap the demonstration file and start on your own databases. The command NEW does just this, and places you at the 0th record ready to receive Field names. The only difference between adding fields now, and adding fields into an existing database (as above) is that you do not need the BY command. So just enter things like: ADD FRIENDS NAME press ENTER, then add the second field name: ADD PHONE NUMBER etc.
Once you have set up the field names, simply enter ADD on its own, and the computer prompts you for the data for the first field of the first record, followed by the second field etc. until the first record is completed. This is then displayed for checking purposes (see previous paragraph to alter data, if required). Start on the second record by entering: ADD etc. etc.

SAVING DATABASES

All the data, system variables, any special words you have created etc. is SAVEd in the normal Jupiter ACE manner just by entering: SAVE name where 'name' is any valid filename. This can also be VERIFIED, and LOADED in the usual way. DATABASE is supplied in this unprotected state to enable you to make as many different databases for your own use as you like. HOWEVER, IT IS ILLEGAL FOR YOU TO MAKE COPIES OF THE SOFTWARE FOR OTHER PEOPLE, AS THIS ACTION INFRINGES THE COPYRIGHT.

MEMORY

The minimum configuration for DATABASE is a Jupiter ACE fitted with a 16K RAM pack, but many applications will benefit by Fitting a larger unit such as the BOLOFIELD 48K RAM expansion. To assist users the software keeps an eye on the amount of memory consumed, and this information is displayed at the top of the dictionary when you enter VLI (a modified form of VLIST). On initial loading with the European data the figure is 20% utilisation - which assumes that a 16K RAM is present. In order to update this assumption when the larger RAM unit is fitted, simply enter : 48 K
This procedure will not need to be repeated after you have SAVEd the dictionary with your own data for the first time.

PRINTING

IF you had all your Friends names, addresses, telephone numbers, and birthdays as a database file, it might be handy to print out address labels for their Christmas cards. You would have seen in SCREEN LAYOUT how to present the data so that only the names and addresses are displayed, and how to insert blank lines between scrolling records (to match up with the spacing between labels). Now all you need is a suitable printer and driving software! The technique is to LOAD the appropriate driving software (For example from BOLDFIELD UTILITIES cassette) before LOADing DATABASE – so that the DATABASE word FILE is still on top of the FORTH dictionary. You would then set the display parameters (SCROLL, TABS etc) and follow this with the printing command (eg: RS232). Now enter the command that would normally produce the appropriate screen display (eg: REP FOR) and the first label would be printed. Pressing ENTER would print the second label etc.
Some printers will require carriage returns to be sent from the computer after every line of data - and this is easily achieved by leaving 1 GAP set, rather than 0 GAP as mentioned in SCREEN LAYOUT.

SECTION TWO - ADVANCED PROGRAMMING

The real power of DATABASE is that you can write your own FORTH colon definition to produce effective 'command files' that control the way the software runs.

This section of the manual assumes the user has read and understood the ACE/FORTH manual and is familiar with the terms it adopts.

DATABASE employs sophisticated LIST processing concepts, enabling data to be held in an open ended LIST of ITEMs with flexible bounds. All the data is stored as a single string compiled into the word FILE, with ASCII code 0 acting as punctuation between records, and ASCII code 1 as punctuation between fields. This hierarchical sub-division need not end here, as any field may itself be considered as a LIST of ITEMs with other punctuation defined by the user.

GLOSSARY OF WORDS AVAILABLE FOR PROGRAMMERS

The following words are supplied to operate on lists:
(p denotes a punctuation character)

ITEM (s,n,p -- s2) returns the nth item of LIST s using punctuation p. If n is negative it is changed to 0. ERROR 99 if n exceeds the number of items in the list.

#ITEMS (s,p -- n) returns the number of items in LIST s using punctuation p

ENDITEM (s,p -- s2) returns the end item of LIST s using punctuation p

BATCH (s,n),n2,p -- s2) returns the batch of items in LIST s between items n1 and n2 inclusive, using punctuation p

ENDBATCH (s,n,p -- s2) returns the batch of items in LIST s from the nth item to the end item inclusive, using punctuation p

-ITEM (s,n,p --) deletes item n from LIST s using punctuation p. s must be in FILE

+ITEM (s,r,p --) inserts a nul item in LIST s using punctuation p, immediately after item n. If n=-1 then it is inserted before item 0. s must be in FILE.

ITEM# (s1,s2,p,mode -- n) attempts to match string s2 with an item in LIST s1 using punctuation p. Returns the matching item number if found, otherwise -1.

The following words are supplied to operate on strings:
(s denotes a string address and length)

$ (-- pad+1, count) reads in a string (including intermediate spaces) from the input buffer to the PAD. To include trailing spaces, terminate the string with a quotation mark (").

$: (-- s) When this word precedes a FORTH comment in a colon definition, the address and length of the comment string is left on the stack, thus allowing comments to be used as string literals.

-STR (s1,s2 -- flag) Alphabetically compares two strings. Flag=0 if s1=s2 flag=-1 if s1<s2, flag=1 if s1>s2.

$! (s1,s2 --) String assignment operator. Stores s1 in s2, but s2 must be within FILE. s2 expands or contracts depending on s1. If s1 is a nul string then s2 is removed from FILE. If s2 is a nul string, then s1 is inserted immediately preceding s2.

TAB (n --) move print position to column n

MATCH (s1,s2,mode -- flag) leaves a true or false flag depending on the comparison made in one of four modes. Mode checks s2=s1, mode 1 checks s2=left side s1, mode 2 checks s2=substring s1, mode 3 checks s2=right side s1.

The following words are supplied to operate on signed double precision numbers and suitable strings:
(d denotes two stack locations occupied by such a number)

DPS (-- addr) System variable holding the number of decimal places to which numbers are being handled. Initially 0. If DPS is set above 0 then all numbers must comply with the revised format, by including trailing 0s if necessary.

->D (s -- d) converts string s into d ignoring commas. If DPS has been set above 0 conversion continues past the decimal point by the appropriate amount.

>$ (d -- s) converts d into string s, including commas as appropriate. A decimal point is included if DPS is set above 0.

D (-- d) reads in a string from the input buffer to the PAD, and executes ->D

D: (-- d) at compile time, if this word precedes a FORTH comment in a colon definition, then the string within the comment is converted into a double precision number and compiled. At run time this number will be put on the stack.

D. (d --) prints d with full punctuation, followed by a space.

D.R (d,n --) prints d right justified in a field n with full punctuation.

D+ D- D= D> 0< DNEGATE 2DUP 2OVER 2DROP 2SWAP U* U/MOD are double precision equivalents of familiar words (refer Chap. 19 in manual)

The following words are supplied to operate over the entire database:

#RECORDS (-- n) returns the number of records in FILE

RECORDS (-- s) returns the address and length of all the data in FILE

RECORD (n -- s) returns the address and length of nth record

R# (-- addr) system variable holding the current record number

F# (-- addr) system variable holding the current field number

FIELD (s,n -- s2) returns the address and length of the nth field of s where s is the address and length of a particular record

RESET (--) makes the 0th record current

SET (n --) makes the nth record current

NEXT (--) makes the next record current

CRECORD (-- s) returns the address and length of the current record. This word must only be used in conjunction with RESET, SET or NEXT.

#FNAMES (-- n) returns the number of field names within a record

FNAME (n -- s) returns the address and length of the nth field

FNAME# (s n) attempts to match string s with a pre-defined field name using MATCH in mode 1. Returns the matching field name number if found, otherwise -1.

The following words are supplied for use in your own colon definitions

THIS is used to let BY and ? know not to look for a string in the input buffer, but to use the indicated string instead.

REPORT controls the intermediate reporting normally seen on the screen during multistage instructions. 0 REPORT inhibits the display, 1 REPORT reverts to normal.

.RF produces the 'n RECORDS FOUND' display.

DEFINING YOUR OWN ROUTINES

It is essential that FILE (the compiled word containing all the data) remains at the top of the dictionary. DATABASE will issue a warning if you have defined a new word above FILE, and refuse to function until you have removed it. There are five empty words already in the dictionary (U1 to U5) intended for you to REDEFINE your own words into. If you need more than five, you will have to initially SAVE the extra words as a separate program, then LOAD this in advance of LOADing DATABASE (so that the extra words are below all the words of DATABASE) and then this larger version of DATABASE can be SAVEd for future use.

In the following examples the colon definitions are printed to illustrate their meaning, rather than as they would appear if you LISTed them.

Here is a trivial example of a new word that will find and display the record relating to United Kingdom:

```
:  UK
    1 MODE
    $: ( COUNTRY) THIS BY
    $: ( United) THIS ?
;
```

remember the space after the opening comment bracket, and when the word has been compiled remember to REDEFINE U1

The next example is a slightly more useful word:

```
:  SELECT
    0 REPORT
    BY NOT ??< ??< #AND
    1 REPORT
    .RF
;
```

now REDEFINE U2

Try: SELECT POP 1,000 5,000 or SELECT AREA 0 500

Here is another useful word that operates on numeric data:

```
:  SUM
    BY D: ( 0) RESET
    #RECORDS 1
    DO
      NEXT
      CRECORD F# @ FIELD ->D D+
    LOOP
    D.
;
```

now REDEFINE U3

SUM works by making a particular field current (read from the input buffer), a double precision accumulator is initialised to 0, the 0th record is made current, a loop begins by making the next record current, the current field of this record is converted to a number and added to the accumulator, and , after all the looping the sum is printed.

Try: SUM POP (646,076) or SUM AR (11,120,968) and even SUM CAP (0)

The last three examples show in stages how a frequently used instruction sequence can be held as a single word, how your own punctuation can separate a Field into subdivisions, and how to combine facilities to produce exactly what you want from selected pieces of data stored in the FILE.

Imagine that the data to be stored is names, addresses, telephone numbers etc., and that you want to produce name and address labels on a printer.

To start with you may have fields called: NAME , ADDRESS1 , ADDRESS2, ADDRESS3 , ADDRESS4 , POSTCODE, TELEPHONE , BIRTHDAY; and the following word would print the labels:

```
:   LABELS
    0 5 PART                .. selects NAME to POSTCODE fields only
    -0 TABS                 .. omits field names, prints on left edge
    2 SCROLL                .. inserts 2 line feeds between labels
    #RECORDS 1              .. sets the loop counter from 1 to end of list
    DO
      I LIS                 .. prints each label in turn
    LOOP
;
```

However, to avoid having so many fields (many of which would be blank because not all the addresses would need 5 lines), you could instead have the entire address in one field with commas separating the data. In this case the Field names would be NAME, ADDRESS, TELEPHONE, BIRTHDAY; and you would need a command to reformat the address back into separate lines for printing purposes.

```
:   REFORMAT
    2DUP ASCII , #ITEMS 0
    DO
      2DUP I ASCII , ITEM TYPE
      CR
    LOOP
    2DROP
;
```

REFORMAT requires the start address and length of an ADDRESS Field in order to work (see #ITEMS). Now, here is a word that uses REFORMAT to produce labels the': are similar to those previously printed, but using the neater record sty e:

```
:   LABELS2
    #RECORDS 1
    DO
      I RECORD 0 FIELD TYPE CR CR
      I RECORD 1 FIELD REFORMAT CR CR CR
    LOOP
;
```

In fact these are slightly better labels, because there is a blank line between the NAME and the ADDRESS!

This pdf was compiled by Steve Parry-Thomas
June 2005

For Jupiter Ace users everywhere

www.jupiter-ace.co.uk